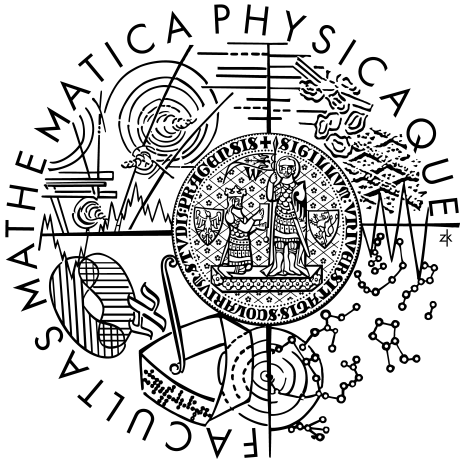
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# DIPLOMOVÁ PRÁCE




*Michal Bečka*

# *Plánování spojů ve veřejné dopravě na mobilních zařízeních*

*Katedra softwarového inženýrství*

Vedoucí diplomové práce: *Ing. Lubomír Bulej, Ph.D.*

Studijní program: *Informatika, softwarové systémy, systémové architektury*

2010

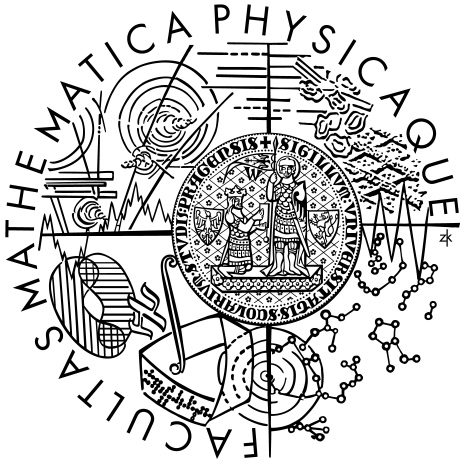
Rád bych poděkoval všem, co mě podporovali během psaní této diplomové práce, především za jejich trpělivost při dlouhých dnech mého psaní a studia materiálů. Zvláště bych rád poděkoval svému vedoucímu Ing. Lubomíru Bulejovi, Ph.D. za podporu a trpělivé vedení mé práce, dále RNDr. Viliamu Holubovi, Ph.D. za výběr tématu a trpělivé vedení v počáteční fázi.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

*V Praze dne 10. 12. 2010* *Michal Bečka*

Charles University in Prague
Faculty of Mathematics and Physics

# Diploma thesis




*Michal Bečka*

# *Mass transport routing*

*Department of Software Engineering*

Supervisor: *Ing. Lubomír Bulej, Ph.D.*

Branch of study: *Informatics, software systems, system architectures*

2010

I would like to thank all that supported me during my work on this thesis for their patience in the long days of my writing and study. I would especially like to thank my supervisor Ing. Lubomír Bulej, Ph.D. for the support and patient supervision of my work, also to RNDr. Viliam Holub, Ph.D. For the selection of the subject and and patient supervision during the initial phase.

I declare that I have written my diploma thesis on my own and with exceptional use of quoted sources. I agree with lending of my work.

In Prague on *10th of December 2010* *Michal Bečka*

# Table of Contents

Název práce: *Plánování spojů ve veřejné dopravě na mobilních zařízeních*

Autor: *Michal Bečka*

Katedra: *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *Ing. Lubomír Bulej, Ph.D.*

E-mail vedoucího: *lubomir.bulej@d3s.mff.cuni.cz*

Abstrakt: *Plánovaní spojů přístupné z mobilního zařízení poskytuje cestovní informace v pohybu. Tato práce studuje různé oblasti plánovačů spojů za účelem implementace řešení pro tuto úlohu. Zkoumáme spojení s telefonními sítěmi a Internetem pro možnosti umístění tohoto nástroje. Dále studujeme vývojové prostředí, složené z operačních systémů a programovacích jazyků. Další část práce pokrývá možné parametry při hledání tras. Dále zkoumáme zdroj a formát vstupních dat o dopravě, spolu s možnými algoritmy pro tento problém. Na základě této analýzy navrhneme aplikaci, pričemž vyhodnocujeme analyzované oblasti. Navrženou aplikaci implementujeme a výsledný program popíšeme a změříme. Cílem je vyvinout fungující aplikaci pro mobilní zařízení zvládající vyhledávání spojů v hromadné dopravě a na ní předvést, co vývoj takového vyhledávače zahrnuje.*
Klíčová slova: *plánovač spojů, mobilní zařízení, cestovní informace*

Title: *Mass transport routing*

Author: *Michal Bečka*

Department: *Department of Software Engineering*

Supervisor: *Ing. Lubomír Bulej, Ph.D.*

Supervisor's e-mail address: *lubomir.bulej@d3s.mff.cuni.cz*

Abstract: *A journey planning tool accessible from a mobile device provides travel information on the move. This work studies various aspects of journey planners in order to develop a solution for this task. We analyze the connection with phone networks and the Internet for possible choices to place the tool. Then we study the development environment consisting of operating systems programming languages. Another part covers possible parameters for the journey search. Then we investigate the source and form of input traffic data along with possible algorithms for this problem. Based on this analysis we design an application, making decisions from the analyzed areas. The design is applied for the following implementation and the resulting program we describe and measure. The purpose of this work is to develop a working application for mobile devices capable of public transit journey search, and to show what development of such application entails.*
Keywords: *journey planner, mobile device, traffic information*

# 1. Introduction

A journey planner is a service that became common in recent years. Since usage of mobile devices spread quickly, it is not surprising these two technologies combined their strengths. Now people can find travel information while actually traveling.

What is a journey planner? Sometimes called a trip planner, it is an application that can find a way to travel by public transport from one place to another at specified time. Apart from these basic parameters, there can be several more, like what type of transport the user wants to use, for example a train, a bus, a tram or just some of them. User can also specify the type of transport in more detail, like the quality of the vehicle, or disability support. If he wants to specify the route, he can add a list of stations he want to pass through and other parameters.

After all the parameters are entered, user submits them and waits for the result, containing the best trip found. He can also choose to modify the parameters of the search, in case he misspelled something, he wants to specify his search or just to find something a little different.

Mobile device is a small computing device, that can fit into a pocket. Usually, a mobile device is equipped with a display for graphic output and a miniature keyboard or a touch screen for input. Different types of mobile devices include pagers, digital cameras, e-book readers, navigation systems, but for the purpose of this work only relevant types of devices will be considered. These are cell phones, PDAs, smartphones and other types of mobile computers capable of running a journey planner tool, or at least accessing it.

## 1.1 Goal

The goal of this work is to develop a a search engine for optimal connection in a city mass public transport for mobile devices. Search engine itself should be independent on the operating system, another part of the work will be an interface for specific environment. The search is to be able to work with static information, like schedules and walking distance, but it should also support dynamic information, like a common delay.

During the development it is also the goal to show what developing such journey planner for mobile devices entails, including the analysis of the problem, design solutions and describing

problems that rise during implementation.

## 1.2 Structure of the thesis

The work starts by analyzing the environment in which the application will run in. That involves an overview of mobile devices, their operating systems and supported programming languages. The possibilities of a mobile device to connect with the outside world are explored to provide options for server communication. Examples of current journey planners are provided to show what progress is happening in the area.

After analyzing the environment, the following chapter focuses on the algorithm analysis. First the algorithm needs transport data to search through, so the data sources and formats are looked into. For the algorithm itself there are many approaches to solve that tasks, so the overview of known algorithms is provided to present candidates for implementation. Journey planner query is specified by search parameters, which are explored in the next part and their usage is compared in existing journey planners. This concludes the analysis.

With the gathered knowledge the next chapter is about the journey planner design. From the previous areas, decisions are made. Data source and format is selected, along with what architecture and data service to use. Programming language for the mobile device is decided, while on the server side the algorithm is chosen with what search parameters it will support.

Based on this design, an application is implemented. The next chapter describes its parts along with the additional decisions that had to be made. The finished program is also measured to show its performance and practicality.

# 2. Environment and architecture analysis

A mobile device usually connects itself either to the Internet, or to another remote location like a cellular network. This provides the journey planner tool with two main locations to occupy. It can be located on the device itself, completely at a remote location, like the Internet, or at both places, splitting the functionality in two parts. When the tool has at least some part of it at a remote location, it has to use some of the data services the device provides. This chapter contains an overview of mobile devices, followed by a look at the programming environment in them. To connect to the remote location, advantages and disadvantages of different data services are also explored.

## 2.1. Mobile device hardware capabilities

Cell phones were the first to emerge, followed by the development of hand-held computers, generally called PDA – personal digital assistants, or palmtops. These two paths soon intersected into what is now known as a smartphone. Today (2010) about 98% of PDAs sold are smartphones [1].

The most important features of a mobile device in respect to the application like a journey planner are the memory size and the network connection. The memory size is important for applications running a journey planner on the device. In contrast applications running on remote servers rely on the network connection. Today PDAs can have even several gigabytes of memory, or at least several hundred of megabytes, with a possibility to add additional memory by an SD card or a memory stick. But devices on the market vary greatly, from these extreme capacities to very little or no free capacity. When there is not enough memory to accommodate a tool, mobile devices still offer data services to remote locations, that can be used by a tool.

Mobile devices are relatively new with rapid development in multiple directions. Before a single mobile device stops working, it can still be used despite being obsolete. That means that the differences in mobile devices in use range greatly. Developers of mobile applications therefore should provide support not just for what devices are best selling at the time, but also for the obsolete ones that are still in use by general population.

## 2.2. Programming environment

Part of the tool located on the Internet server can be programmed in any language of choice, as servers are not restricted by hardware limitations like mobile devices are. But the tool part on the mobile device, if there is any, must make use of what is available. There are many manufacturers of mobile devices supporting applications, including Nokia, Motorola, Blackberry and Apple. More important then manufacturer of the device is the underlying operating system. Different operating systems in turn support different programming languages and application restrictions. This chapter will cover the differences an propose the best language to use.

### 2.2.1. Notable programming languages

***Java ME***

Java follows the philosophy of "write once, run everywhere", to provide the best compatibility across wide range of devices. It is achieved by implementing a layer between the programs and the operating system called Java virtual machine. This layer hides all the differences between various systems and instead introduces an unified environment. That makes it possible for one program to run on various operation systems, provided the Java virtual machine is installed.

It is easier to hide all the differences of desktop and laptop computers, but when it comes to mobile devices, their severe hardware constraints make it impossible to support the full features of the Java environment. For this purpose there is a cut down version of the Java environment, a subset of its features called Java ME. When we look at the versions of Java, there are three basic ones – Java EE – enterprise edition, the most comprehensive one, that enables powerful features (for servers and such). Then there is the SE – standard edition, that provides the standard features and is the most widely used, will full features one can expect form normal PCs. The last is the Java ME – mobile edition, designed for mobile devices. Java ME provides a subset of features found in the SE, and that in turn provides a subset of features in EE. That should guarantee, that the applications written in the Java ME subset will run in the super-set versions. However, there are some features that are specific to the mobile environment, so Java ME contains some features, that are not found in the SE version. Using these features will make the application incompatible with the SE, but that is necessary only

for the part of the application specific for the mobile device anyway. The general features, those that are found in the SE version too, are the same, there are no different versions of it. This makes the application compatible, apart from the mobile specific part. The application logic can be safely run in the SE version, just with some changes in the mobile specific code. In short, what can be compatible, is compatible.

The main difference from the virtual machines on normal PCs lies in configurations – CLDC for less powerful devices, or CDC for more powerful once. A configuration describes the basic set of libraries and features a virtual machine on the device has. On top of this layer sits another layers, on CLDC sits MIDP – rich set of Java APIs for the use by applications. For more information see [2].

***C/C++***

C and C++ family of languages. The main difference between different devices is not so much in syntax, but in the available libraries and APIs. Often device manufacturers provide their own libraries, sometimes even their own modifications of the language. Thus they lack the compatibility of the Java ME environment. On the other hand, they can provide access to device specific features, that would not have been available in universal API.

### 2.2.2. Overview of mobile operating systems

Notable operating systems, listed by a market share of smartphones sold in the third quarter of 2010 , according to a study by Gartner [3]:

1. 36,6% – Symbian OS – Currently used by Fujitsu, Nokia, Samsung, Sharp, and Sony Ericsson, in the past also by BenQ, LG, Mitsubishi and Motorola. Currently being succeeded by Symbiam platform, which is fully open source. Programming languages include C++, Java ME, OPL, Web/WAP scripting.
2. 25,5% – Android – Used by Google. Supports Java ME.
3. 16,7% – iPhone OS – Used by Apple Inc. Main programming language is Objective-C.
4. 14,8% – RIM BlackBerry OS – Used by Research In Motion in their BlackBerry devices. Supports Java ME.
5. 2,8% – Windows Mobile, Windows Phone – Used by Microsoft, these systems are part of the Windows Embedded family [4]. Development tools are available at [5].

Programming languages include Visual C++, Visual C#, Visual Basic.

6. 2,1% – Linux – Used by Motorola in China, by DoCoMo in Japan.
7. 1,5% – Other operating systems.

## 2.3. Data services

This chapter explores the data services a mobile device can use to connect to the outside world.
First let's show the data services that have a potential to be used by the tool. In addition, a data service will be listed in a role it would take in the tool architecture. If it would be used only as a presentation layer with no tool part on the mobile device, or if it would be used as a means of communication between two parts of the tool, partially hidden from the user.

***SMS service (presentation layer) –*** Originally part of the GSM standard, it has expanded into other mobile technologies. An SMS service is now the most widespread way to send short text messages from a mobile device. A tool must use additional service, an SMS gateway, to handle the messages between itself and the user. This allows the tool to be placed on the Internet.

***Email (presentation layer)*** - An Internet text message service. Both email and SMS are text messaging services, but the main difference between email and SMS relevant to the tool is that SMS connects to the cellular network, while email connects to the Internet. Therefore there is no need for an additional service like the SMS gateway.

***Web page connection (presentation layer)*** – A service allowing the tool to be completely placed on the Internet, while it is accessed by a web browser on a mobile device. With web browsing capability naturally comes standard internet connection, so the web page browsing does not have to be considered from the internal communication angle. For less powerful mobile devices a web page access is often done through WAP – Wireless access protocol. It allows the mobile device to connect to a WAP gateway that acts as a proxy between wireless network and the Internet and is transparent to the user. WAP browser works with WML – Wireless markup language, adapted for the lack of resources on a mobile device. WAP

gateway translates HTML content into WML. Notable alternative to WAP is the Japanese i-mode. More powerful devices use XHTML MP, or even full HTML.

***Internet connection (internal communication)*** – A pure IP based connection is ideal for the internal tool communication, enabling communication between tool parts in its own native way. Since „2G" networks, General packet radio service (GPRS) is available to users, allowing IP protocol access to the Internet.

***No data service*** – The tool would be located completely on the mobile device.

An adaptation of the tool would be to use a voice communication with either a live person handling search queries, or a voice recognition.
Other services may include MMS – a service allowing to send multimedia messages, but such service is not suitable for specific data transmissions. At most it can be used to provide a nice looking search result, but it is not suitable to send any information from the user.
While it is possible to use different data services at once, one for transmitting and other for receiving, it would only introduce unnecessary complications and compatibility issues. Therefore it is presumed that the tool uses only one data service for a single search.

## 2.4. Existing journey planners

With the widespread use of the Internet, journey planners gained popularity. Nearly every transport operator provides some kind of journey planning, either for its own services or combined with some of its competitor's, so listing all of them would be impossible. Instead only notable examples will be shown.

A journey planner is not to be confused with a route planner. Example of a route planner would be a car navigation. The difference is that a journey not only covers a route, but also time of travel, taking into account individual transport connections and exchanges between them. A route is just a connection of places to go through.

### 2.4.1. Google Transit

Google Transit [6] is becoming the most widespread journey planner tool in the world. Launched in 2005 as a part of Google Maps, its goal is to cover journey planning all over the world. Despite being relatively new, it has grown much, mainly because it is backed by the corporate giant the Google is. What also made it popular is being incorporated into an existing technology, the Google Maps, allowing the user to work with it seamlessly. In classic journey planners user works with them as with a standalone application, while integration in interactive maps can provide several search parameters automatically, making it easier to use.

Architecture of Google Transit it tied to the Google Maps. Originally only available on the Internet, in 2006 Google Maps for mobile was released. First it was intended to run on Java-based phones or mobile devices, providing many features of the web site, but in time it was adopted to other platforms. Finally in 2007 the Google Transit feature was added to Google Maps.

Of the discussed architectures it fits the Internet connection, having part of the tool on the device and part on the Internet, connected most likely by an internal Internet connection.

Google Maps are supported by every major mobile operating system, including Symbian OS, RIM BlackBerry OS, iPhone OS, Windows Mobile, Android and others.

### 2.4.2. Transport Direct Portal

Transport Direct Portal [7] is a comprehensive journey planner in the United Kingdom, covering England, Wales and Scotland. Many modes of transport are combined into this tool. In 2005 it introduced access from mobile devices. From the architecture point of view, it is accessible as a website, customized to be viewed from a mobile device. Setting up on the mobile device involves at most creating a bookmark. Since there is no part of the tool on the mobile device, the compatibility remains wide, available wherever web browsing is available.

### 2.4.3. Idos

Local to Czech republic [8], it provides journey planning for many modes of transportation, including trains, buses and city public transports. The Department of transport payed for its development and now it is managed by the CHAPS company. It does not have any notable competitors in the country.

Although it is possible to view the web page from a mobile device, a notable architecture solution is a standalone application [9] that can be run from a mobile device. It is unusual that the application is not free of charge. The installation of the application itself is free, but for the use of the data files with transit information user has to pay a periodical license fee.

The only operating system to support the standalone application is Windows Mobile. In the past it was available for Pocket PC, but the support was discontinued.

### 2.4.4. Comparison

Provided examples were chosen for their significant differences. Google Transit is an example of a global tool with wide area coverage, using the services of both a mobile device and the Internet as much as possible, backed by a powerful commercial software company. Transport Direct Portal is an example of a service with significant cooperation of different areas of transport, focusing mainly on the journey planner itself on a web page, with mobile access not being its main priority. It is backed by a nation and is following significant research on the subject. Idos is a journey planner for local use with little cooperation between areas of travel, enabling it to be custom-tailored to its tasks. Mobile device standalone application does not seem to be a serious effort for a mobile device tool.

# 3. Algorithm analysis

The search algorithm is the core of the journey planner. It receives two inputs, transit data and search parameters, providing the optimal result found.

## 3.1. Public transit information

In order to find a journey between stations, apart from search parameters the planner needs actual data to search through. Such information includes list of stations, connections between them and traffic schedules. This chapter will cover where the data comes from, in what form and how hard is it to get it.

### 3.1.1. Data sources in general

Each transit company keeps records about its traffic schedules and much more. Traffic schedules are just a top of the iceberg. Basic records include vehicle inventory, employee records, list of stations, service depots, tracks and all the equipment. With this data there are task to do like the design of transport routes, design of public and service schedules, assignment of vehicles to individual connections or assignment of personnel to shifts and vehicles. The final schedules available to the public are result of a long and careful planning. Even though such data are generally available in forms like printed schedules, application ready data are often withheld from the public. Transit company provides them to selected sources carefully, since they are a valuable commodity for often commercial journey planners. Even if a journey planner is free of charge, it might generate profit through advertisements or by other financial support, for example by national transport department funding. On the other hand, making the information available to successful journey planners promotes services of the transport company, encouraging it to share the data. Licensing issues often accompany publishing of the data.

Google is approaching the data gathering by providing its data format GTFS and letting transport companies provide the data themselves through Google Transit Partners Program. This approach seems to be working, for as of 5. December 2010, the program reports 213

transit agencies from all over the world providing their data.
Transport Direct, even while covering only the United Kingdom excluding Northern Ireland, also acquires its information from many different sources depending on different transport modes. An example may be Traveline – for buses, tram, light rail and ferries, TheTrainline – for train information, East Coast and others.
In the Czech Republic, one company – CHAPS, was chosen by the Department of transport to manage national information system of traffic information. All transit companies are required by the law to provide their traffic schedules to this company, aside from a few exceptions. CHAPS provides many services like the Idos journey planner, some transport coordination and management. The traffic information gathered is available for a certain fee.

### 3.1.2. Transmodel

There are many transit operators all around the world. In the past nearly every company had its own conventions on how to store the traffic data, making cooperation between them difficult. The first serious attempt to standardize the format was Transmodel [10]. It was established in 1992 between several European countries and has been evolving ever since. It is a standard from the European Committee for Standardization (CEN), that provides reference data model for all public transport information. It covers a lot of public transport concepts like scheduling, fares, driver rosters, vehicle planning, and most importantly, journey planning. Concepts are described by Entity Relationship Model and UML and described in detail, providing an unified way to represent the public transport data. One other feature is establishing a clear terminology, since many operators have a different view on what many terms mean, like a trip, journey, service journey and a route. Misunderstanding these terms can lead to compatibility issues not just on code level, but also in research papers and publications.
Transmodel is very generic in describing concepts, leaving specific implementation on readers. This provides sufficient freedom while ensuring compatibility between individual Transmodel inspired systems. This freedom led to more specific standards, like TransXChange, which is a XML standard used in United Kingdom for sharing bus timetables.

### 3.1.3. GTFS

Originally called Google Transit Feed Specification, it was created as a common data format for information supplied to Google Transit. However over time more applications started using this format, with many transit companies sharing their data between each other in it. Its widespread use led to the replacement of the word Google, making it General Transit Feed Specification.
Unlike Transmodel and its standards, GTFS defines very simple and specific data format for schedule information and no more, only what is necessary for journey planning, without concern about internal company data management.

### 3.1.4. JDF

As was mentioned before, transport companies in Czech Republic have to supply traffic information to a national information system of traffic information. JDF is a data format mandatory for this data. In scope it is similar to the GTFS, it includes only information relevant for journey planning. The main difference is that the format closely resembles that of the printed schedules, containing information about train and bus properties like disability support, additional space for luggage and bicycles, presence of a dining car and seat reservations.

## 3.2. Real time information

In addition to static information there are some dynamic factors to consider.
First are changes in traffic schedules. Some are planned, like track repairs, other are unexpected, like accidents. Both can be handled the same way static information is, by updating current data.
More interesting is real time information influencing traffic. This includes estimation of current vehicle position, current delays and traffic congestion for bus routes. For this purpose there are data standards, enabling journey planners to retrieve such information as well. One such standard is SIRI - Service Interface for Real Time Information, for sharing real time

delays and timetable updates among other things. It is based on Transmodel, which in its thoroughness covers even this issue. Sources of this data are usually centers of traffic operations and control.

Integrating dynamic information with static information is the easy part. Before feeding data to the search algorithm, static and dynamic information is combined and the result is passed on.

Many public transport companies use automatic vehicle location (AVL) systems to keep track of their fleet. Such systems, usually based on GPS technology, transmit location of vehicles in certain periods of time. This enables predictions based on a real-time location of vehicles to be added to the static schedule data.

Specific examples of real-time data sources can be NextBus [11] in San Francisco, which accepts queries and returns real-time data in XML format, CTA Bus Tracker [12] in Chicago, or OneBusAway [13] in Seattle.

When a journey planner combines different modes of transportation and different transport operator services, it can have real-time information from different sources. [14] describes a routing application for a road network, and although it is not about public transport, it provides an interesting model for combining different sources of real-time data. For a road network sources could be local radio station, official information from ministry of transport or local weather data. For public transport, such information is not relevant, for schedule correction or real-time data will be published only by the operator providing the transportation service. But by combining different operators and unifying their data, multiple sources of real-time data also have to be unified. [14] proposes a central server to plan with static schedules, combined with distributed architecture of real-time data gatherers based on a middleware.

The processing of location information is not instantaneous. There are delays in data collection, processing at the central server and in publishing that information. [15] researches these delays an proposes a way to reduce the error. The results are implemented in a real life transport company in Korea, showing 23% reduction of error in prediction times. Therefore when combining the real-time data, the delays should be considered and minimized.

Before the real-time location information can be used in a journey planner, it has to be converted to time point data. This prediction of arrival and departure times based on the vehicle location can be done by the operator or by the journey planner. It depends on what

information does the transport operator publish.

There has been research in this kind of prediction, an algorithm for such prediction can be found in [16]. For the prediction, not just current locations of a vehicle is used, but also a historic data about previous delays of a vehicle on the same route. More general research in [17] classifies different kind of predictions independent on the prediction algorithm and measures predictability of the data. Interesting results are that the predictability is low, predicted values are inaccurate beyond the near future, for that static prediction (limited to scheduled data) is the best predictor. Also the prediction based on previous delays on the same route segment was more accurate than prediction based on real-time data of the vehicle on previous route segments, except when the delays were very high. The paper concludes that the prediction errors are highly data-dependent, suggesting that different prediction algorithms might be needed even for different routes.

## 3.3. Existing algorithms

There are many algorithm to solve the journey planning problem. However, it is a problem that is connected to other problems like what search parameters to choose and what traffic data are available, which blur simple distinction between the algorithms. So for the purpose of this journey planner, several algorithms were chosen that represent the main categories.

### 3.3.1. Headway algorithms

Journey search combines a location aspect, searching for a route, with time aspect, that is the time dependency of scheduled services. One approximation to deal with schedules is to ignore them and compute only a headway. That is how long in average it takes for the next transport vehicle to arrive.

### 3.3.2. Brute force graph search

A simple solution is to use a graph search algorithm where nodes would be the location

combined with a scheduled event, like a departure at a stop or an arrival. Search boundary would be the time, nodes would be picked chronologically and the best way in that node would be computed based on its time and location based predecessors. A* improvement can be added, using the maximum speed of any transport vehicle in the network, that number is possible to compute during pre-processing.

### 3.3.3. Transfer matrices

The algorithm from [18] prepares transfer matrices during pre-processing. Transfer matrix $Q_k = [q^k_{ij}]_{i,j=1..n}$ $(k = 0, 1, ..., maxt)$ is a two-dimensional matrix where *maxt* is the maximum number of transfers and $n$ is the number of stops. $q^k_{ij}$ equals 1 if there exists at least one route from stop number $i$ to stop number $j$ with exactly $k$ transfers. Additional matrix $D = [d_{ij}]_{i,j=1..n}$ is created during pre-processing, which contains the minimal number of stops on a route from $i$ to $j$. Templates of optimal paths are created from the transfer matrices. A path template is a route containing only possible transfer stops. That concludes the pre-processing part.

After receiving a query, graph searches are initiated from the departure stop node. When going through the network, path templates from the departure stop to the destination are filled with specific time data, for example what buses to take at what time and how long will the journey take. Since this includes a lot of paths, an approximation is used using the $D$ matrix, similar to A* heuristics, resulting in only a portion of the paths fully searched.

The search itself is recursive. Starting from the initial node, candidates for the next stop in a path are selected using the path templates. Time arrivals at the candidate next stops are computed and those with more than the minimal arrival time + a tolerance $\varepsilon$ are excluded. Also, using the $D$ matrix approximation, the only those candidates with minimal number of stops to the destination are considered. Now, with one of a few possibilities of a next stop on a path, the search step is performed recursively on them. The recursion ends when the destination is reached.

This recursive search is performed once for each number of transfers, working only with the path templates of the corresponding length. So first the direct paths are searched, than paths with one transfer, and so on. This way, the depth of the recursion tree is always the number of transfers considered. The result of those searches is a set of complete paths. The last step is to choose one of them based on total travel time or number of transfers.

### 3.3.4. Label setting algorithm

This algorithm from [18] is used to find *k (for k ≥ 1)*, sub-optimal paths. Pre-processing involves creating the $D = [d_{ij}]_{i,j=1..n}$ matrix of the minimal number of stops on a route from *i* to *j*.

To find different paths through the network, it is possible either to delete some edges, or use node labeling, which this algorithm chooses. A label at a node specifies how many paths were found to it from the departure node. During the graph search, all the paths found are remembered by creating a new edge for each of the paths. This way the traversed graph is one edge away from the departure node. The number of paths found to each node is limited by *k*.

A search step of the algorithm finds nodes with less then *k* paths found with the minimal time stamp + a little tolerance *ε*. From these nodes, one is selected that has the minimal number of stops to the destination using *D*. The new paths found are from the destination through the selected node to its neighbors, just like relaxing during Dijkstra [19]. The selected node's neighbors have their label incremented and a new edge from the departure stop to them created. The algorithm stops when the destination node's label reaches *k*, meaning that there are *k* paths found from the departure stop to the destination.

### 3.3.5. Pattern-first search

In [20], during pre-processing, the service lines on the network are reduced to patterns, which is a path containing only possible transfer stops. Unlike the path template in one of the previous example, which used different service lines between its transfer nodes, a pattern corresponds only to one service line with nodes where the traveler can make a transfer to a different service line.

Before each query, an active part of the network is created. It consists only from the patters of lines active in the specified time, so for example in a work day the algorithm does not have to search through the weekend lines.

A single search is initiated from the departure node, similar to Dijkstra [19]. The step of the search finds an open node with the minimal time stamp. On all the outgoing patterns from this

node, minimal arrival time on all the remaining stops is updated. The updated time on the patterns is the time it would take there using only that pattern without transfers. Than the selected node is closed. The algorithm ends when the destination node gets closed.

### 3.3.6. An algorithm for finding reasonable paths in transit networks

In [21], the pre-processing part prepares the network independent on schedules, just connected locations. For each location, a list of adjacent nodes is compiled.
Like in the algorithm with transfer matrices, path templates are used, here called via-node lists. But not as part of the pre-processing. When a query is received, a recursive search is started from the departure node to find all via-node lists to the destination. Number *delta* is used as a maximum number of transfers to limit the depth of the recursion. A transfer can be a standard line exchange at a stop, or a walk link between close stops. After all the via-node lists are completed, specific time values are computed for them, according to what service lines can a traveler take on that path. After the paths are completed, a minimum total time of a path is computed. All the paths that take longer than the minimum plus a parameter *tau* are excluded. Those left are considered “reasonable”.

## 3.4. Transit search parameters

When a person requires a service providing travel planning, the end result should be to provide sufficient information for him to follow. To achieve this the planner first needs some input. This chapter will have a look at what a typical user would require. Those parameters will be evaluated later when studying how well they can be implemented by a search algorithm.

### 3.4.1. Departure and destination

When looking for a route in transport, user usually has a clear idea of the origin and the destination of the trip. In this application, as well as in most of the journey planners, it is not possible to choose any place at will. A valid place is usually a transport station, with

connections to the transportation network. Furthermore this station has to be known to the application and it has to have the data about it and its transport connections. But this is the concern of how complete the data are and not the immediate concern of the user. So the data about stations should be as complete as possible and the only thing user has to think of is to identify the closest stations to his departure and destination places.

There are ways these parameters can be expanded though, using different services. Such as finding a connection between places other than transport stations or giving an estimate of the time to get to the first station. That would however require additional services, like information about roads, or GPS coordinates. Complex applications like Google Transit provide this information though. As it is part of Google Maps, it has access to roads and other ways of travel, so it can chart a route from almost anywhere. Even if a tool does not support these external services, it is a good idea to make the journey planner prepared for them, so they can be easily incorporated in the future.

These parameters sometimes does not have to be entered, but sometimes they can be supplied by external services. A mobile device provides a possibility of supplying the origin based on the location of the device. Google Transit provides this service under the name My Location. It does not even have to support GPS, it can estimate user location based on unique footprints of nearby cellphone towers providing reception to the device.

When user is not looking for a specific destination, but for example for a nearest museum, other service can provide the location, which can be supplied directly to the planner.

Proposed parameters:

- Origin
- Destination

### 3.4.2. Time

Another basic parameter to define the trip is time. There are two main ways how to define it, by the departure time or the arrival time. Unlike places however, it is not so straightforward. It is highly unlikely that the time user specifies will be the exact time of either departure or arrival, so it cannot be fixed like places are.

The ways to specify the time are several, like looking for a connection departing or arriving before or after a specified time. To decide which ones to implement, first consider the

situations user is in when he uses them.

- The first most likely case, user wants to get somewhere and be there at a specified time. This means to search for connections arriving before specified time.
- The another most likely case, user want to get somewhere as soon as possible. That means to depart after specified time, with the specified time being the first moment he can.
- Another case, user want to arrive somewhere after specified time, when for example an accommodation or another service is made ready for him at that time.
- The last combination of before/after and departure/arrival time is to depart before the specified time. That can be the case opposite of the previous one, when an accommodation or another service is no longer available.

The first two choices are the ones used by all the journey planners. The last two ones are interesting, but the necessary result can be obtained by traditional search by either by making a query with a different time, until a satisfying result is returned, or in case there are several consecutive results returned at once, by just selecting a convenient result from the list.
Like an origin of the trip, time parameter has its default value. Current time can be set as th time of departure, specifying immediate travel query.
Proposed parameters:

- Time.
- Specification if the supplied time is a time of arrival, or time of departure.

Now we have all the basic information needed to produce search results: The departure place, the destination place, and the departure or destination time.

### 3.4.3. Trip duration

Without any more parameters, the obvious way to judge the quality of the desired connection is the time it takes. But there are several more things that can determine the quality of trips. For example it is the number of exchanges on the way. There are also other parameters that can influence the result, like restrictions on the type of transport or the delay expected. In the rest of the chapter we will look at these parameters more closely and how are they related to each other.

The most important property of the trip will be its duration. While other parameters do not have to be optimized, this one always will. The shorter the trip, the better. However, other parameters can sometimes be optimized at the expense of this one. Since it is always optimized, no additional parameter is necessary.

### 3.4.4. Number of exchanges

Another parameter is the number of exchanges. A connection will consist of a trip from the departure to the destination, and it does not have to be by one vehicle only. Different parts of the journey can be traveled by different vehicles, with necessary exchanges on the way.
One way to limit the number of exchanges it to specify the maximum, so any trip with more than that number will never be considered as a valid result. This has a disadvantage, that there is no optimization among trips that fit into this criterion. Other way is to try to minimize the number of exchanges. This optimization would work between all the trips. However, it is much harder to define how aggressive it should be, since this would definitely work at the expense of the total traveling time. How to choose between a shorter trip with more exchanges and longer trip with less exchanges? One way to do it is for the user to define, how much time he is willing to sacrifice in favor of one less exchange.
Proposed parameters:

- Maximum number of exchanges.
- Number of travel minutes to sacrifice for one less exchange.

### 3.4.5. Exchange duration

Exchanges also take time and this time is another parameter to consider. Some of the travel planners let the user define the minimum or maximum time for the exchange to take place. The maximum may seem irrelevant, since the application will try to minimize the time of the trip including the time spent by exchanges. So by setting a maximum for an exchange will make the searcher choose another trip, where user will spent more time in the traveling than he would save by the shorter exchange. However, sometimes no result returned is better than returning bad results. For example, when searching for connections in the evening, the

maximum could exclude trips with exchanges waiting until the next morning.
The true reason for setting the maximum is probably to help algorithms to limit the number of possibilities to search through, because most users would welcome the extra info about the overnight connection and deduce from it that there is no better evening result.
One time limit for all exchanges at once may not be desirable. Some exchanges can take much longer that others, for example from one train to another it is much faster than from one train to an airplane. There is a possibility to specify different limits for different properties of vehicles or station. But that information would most likely be provided by a transit company with traffic data, not by a user.
Proposed parameters:

- Minimum exchange duration.
- Maximum exchange duration.

### 3.4.6. Exchange location constraints

Apart from time constraints for exchanges, there are also place constraints to consider. In that case the exchanges would be limited to specified places. It wouldn't make sense to require exchanges at all of them, since straight train would be faster. And if the user wanted to stop at all of those stations anyway, it would mean that he does not just want to go from the departure point to the destination, but to visit other places, and for that more separate trip searches would be more appropriate.
Also the place constraints do not have to be limited to exchanges, user just might want the route to go through specific places. This is handy when the searcher provides unwanted routes that are not familiar to the user, or otherwise inconvenient.
Opposite way is to specify what stations the user does not want to go through.
Proposed parameters:

- Stations where only at them can the trip make an exchange.
- Stations the trip must pass, independent on if there is an exchange or not.
- Stations where where not to make an exchange.
- Stations the trip must not pass.

### 3.4.7. Walking

Sometimes it is possible to walk between close stations. This form of transport is somewhere between an exchange and another transport mode, typically used between stations that are too close to each other for other means of transport. It is desirable to select user's average walking speed, so there is enough time for an exchange for an elderly person, or that there is not much time wasted if user is in a hurry and doesn't mind to run. One more advantage of walking is that in some rare cases it would be faster to walk somewhere than to use public transportation, probably because the public transportation can take a long detour.
Proposed parameters:

- Maximum walking distance user is willing to traverse.
- Average walking speed.

### 3.4.8. Reliability

Transport connections are not always reliable. The projected trip might not be available when the user follows it. When deciding the reliability of a trip, let's first have a look at how a trip can fail, what are the sources of unreliability.
The first one would be a missed exchange. This can be caused by a delay of a previous connection, or when the next connection would not go at all, because of some accident or a breakdown.
Other thing to consider is a delayed arrival of the whole trip. Unlike delays between exchanges, this concerns a delay of a vehicle after the last exchange. Although it does not share the danger of a missed exchange, there may be other obligations user can have after the trip, that can also make these delays part of a reliability property. But the end result is satisfied, the user ended in the destination, even if late, so for the rest of the work it will be ignored.
So now we have established that the main reliability issues are in a failure of an exchange due to delays and a failure of a connection due to breakdown. These two values can be represented by a probability value in percents.
The trip does not just have to succeed of fail completely. When something happens, it may be possible to avoid the obstacle by taking an alternate route to the destination. That too

influences reliability. When there are more alternate routes to the destination for some parts of the journey, naturally the trip is more reliable, than if there was just one connection available. However for the purpose of this work we will consider only two outcomes, success or failure.
When the connection fails and another route must be taken, it is possible to enter another search query to find the route. It does not have anything to do with the search algorithm itself, this functionality can be completely covered in the user interface, in the presentation functionality. If the user can just enter the new search, it does not have anything to do even with the application itself. But there may some support in the user interface to ease this, so the user does not have to enter all the parameters again.
So, how the user should be able to specify the level of reliability? Since it is about probability, it can be in percents how much does he want the whole trip to be reliable. Another way is to specify reliability of each exchange in percents. It lacks the elegant nature of the whole trip number, but it is a good way to avoid spikes in unreliability when the normal risks are not what concerns the user, just a presence of an increased risk.
Alternative to fixed limits is to provide more flexible constraint tied to the traveling time. It would be how much longer the user is willing to travel for a unit of reliability, for example how many minutes he would sacrifice at an exchange for a percent in reliability.
To make it even simpler, there can be one single time variable specifying the minimum time for a reliable exchange.
Proposed parameters:

- Minimum reliability of a whole trip in percents.

  or
- Minimum reliability of each exchange on the trip.
- Number of travel minutes to sacrifice for a percent in reliability of one exchange.
- Number of minutes it takes for an exchange to be considered safe.

### 3.4.9. Trip duration

One would think that after specifying the time of departure, the concern of the application would be to find the trip that will take the least time to get there. However, duration of the trip itself is not the whole time variable that can be considered. The other part is the time between the user specified time, to the actual departure/arrival. For example, user wants to find the

best connection after the time 6:00am.

The first connection to consider departs at 7:00am, and arrives at 10:00am. The other connection departs at 9:00 and arrives at 11:00am.

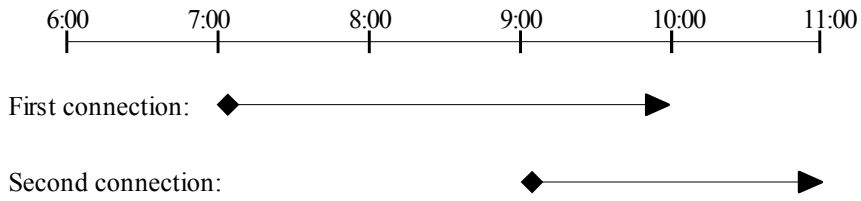



*Diagram 1: Trip duration conflict*

The shortest duration is the second one, taking only two hours. But the first connection, despite taking an hour longer, arrives to the destination one hour sooner. Since the user wanted to get to his destination as soon as possible since the time 6:00am, it makes sense to choose the longer one. I this case what the application is really trying to minimize is the time it takes from the specified time to arrival. So there is a question whether the duration of the journey is to be minimized alone, or is the delay between the specified time and the duration of the journey also the subject of minimization.

One way to avoid this is to have the departure or arrival time bound between two times, minimizing the other parameters like before, but the duration of the trip would be no longer dependent on specified times other than being between them. This could be beneficial when the user is busy and has to get somewhere, so he needs to waste as little time as possible by traveling.

Proposed parameters:

- Second time parameter, with the intent to find the shortest trip between the earliest departure an the latest arrival time.

### 3.4.10. Transit specific properties

Both stations and vehicles can have some properties. It can be their quality, safety, services and more. Those properties can be different for different transit companies and countries, so when making an universal travel planner, they have to be supported but cannot be specifically defined. Working with such properties can be difficult, so let's see what influence they can

have.

When the property is a service, or a type of quality, user might want to restrict the trips to provide those services. So the property can be a restriction on what vehicles or stations the trip can use.

Properties do not have to influence the search at all. Some may be there just for informational purposes, that could be displayed with the result. That could be the presence of a dining car in the train and a disability support. These properties can be specified as restrictions as well, but do not have to be.

## 3.5. Search parameters on existing planners

Part of analyzing the search parameters is to compare them with the ones being commonly used. Selected example journey planners were listed in the environment and architecture chapter.

### 3.5.1. Parameters available on the mobile device

Some parameters are common to all of the listed planners:

- Origin and destination.
- Time.
- Specification if the supplied time is a time of arrival, or time of departure.

These are all the parameters Google Transit and Transport Direct provide in their mobile version. Idos, being a standalone application provides additionally:

- Maximum number of exchanges.
- Minimum exchange duration.
- Maximum exchange duration.
- Stations where only at them can the trip make an exchange – in this case just one station.
- Stations the trip must pass, independent on if there is an exchange or not – again only one station can be specified. This parameter is mutually exclusive with setting

exchange location in the previous parameter.

- Number of travel minutes to sacrifice for one less exchange.
- Number of minutes it takes for an exchange to be considered safe.

### 3.5.2. Parameters available on the web page

On the web page it is possible to fit much more search parameters than on the mobile device. And since many new mobile devices can view web pages in the same way normal computers can, web page search parameters will be considered as well.

Google Transit offers the smallest amount of search parameters, the search on a mobile device and on a web page offers the same features.

Transport Direct offers much more parameters:

- Transit specific property – type of transport, train, bus, underground.
- Maximum number of exchanges.
- Minimum exchange duration.
- Stations the trip must pass, independent on if there is an exchange or not – only one station can be specified.
- Average walking speed.

Idos additionally offers:

- Stations where only at them can the trip make an exchange – on the web page three stations can be specified instead of one.
- Stations the trip must pass, independent on if there is an exchange or not – again three station can be specified. As on the mobile device, user cannot use this parameter at the same time as the previous one specifying exchange places.
- Maximum walking distance user is willing to traverse.
- Transit specific property:
  - Type of transport, train, bus, city public transport.
  - Train quality restriction.
  - Presence of a sleeping car on a train.
  - Option to prefer frequented connections.

It is interesting that the Idos web page lacks some features of the mobile application, suggesting that they are much different from each other:

- Number of travel minutes to sacrifice for one less exchange.
- Number of minutes it takes for an exchange to be considered safe.

# 4. Design

So far the work contained an analysis of the areas and problems the transit planner has to face. This chapter will evaluate that information, to design what features the developed transit planner will have.

To start with, the journey planner has to select a data source, which will provide the transit information. Another step is to select an architecture of the application, if it would be a client-server application and where will different logic parts be located. The choice of a preferred data service of the mobile device to connect to the outside world will influence the architecture decision, together with the requirements of handling the traffic data. The original goal was to implement an interface for a specific environment. In case this would involve a mobile device application, there is a choice of the programming language to use.

From the provided search algorithm candidates one will be chosen for the task. With algorithm known it will be possible to evaluate which search parameters will be supported and which will be unsuitable for implementation.

## 4.1. Data source

### 4.1.1. GTFS

From the candidate data sources, the GTFS seems the best choice. The first major advantage is its availability. While other formats generally are for internal use only, the GTFS data is publicly available for download from a web site. This central point enables the journey planner developers and transit companies to efficiently share data. The second advantage is its widespread use. By using the published data in a journey planner included in one of the most globally used maps, Google managed to encourage the transit companies to share their data voluntarily for free. Another advantage is its universality, by supporting transit data from all over the world and hiding company specific information, a journey planner based on GTFS will be much more universal than when using other formats.

### 4.1.2. Real time data

The usefulness of real time data is researched in [22]. Based on an experiment in a corridor from the Massachusetts Bay Transportation Authority, it measured that the actual time savings from the path decisions based on real-time data i negligible. However, it acknowledges that the usefulness depend on how much the actual travel times differ from the scheduled ones. In less reliable networks, real-time data value increases. Also the usefulness may increase greatly when there are not just delays, but breakdowns that make a route completely unusable.

Apart from path decisions, real-time information has additional benefits. By providing the user with more precise estimations of arrival and departure times, it eases the uncertainty during waiting on a stop, improving the user's trust in the service. That in turn will improve the odds that the user will choose to travel the public transport at all, not by a car. With the increased congestion problems in cities, more people using public transport benefits the city, the passengers and the transport company, that usually develops the journey planner. Therefore, there is no reason for a transport company not to include the real-time information in their journey planner, apart from the development difficulties, which for the transport company should be negligible.

Based on this information, the developed journey planner will have a single universal API, that will be able to receive corrections of the scheduled data it already possesses. For each source of real-time data, there can be a module, that will translate the source's format into the universal API. The modules can use any means to gather the data, including a middleware, independent on the journey planner. If the received data is already in form of predicted schedule adjustments, only the format change would be necessary. If the data consists of vehicle locations, the module will have to employ a prediction algorithm to convert the data to schedule adjustments. This approach enables different sources to use custom tailored prediction algorithms, which follows the findings of [17] that the prediction errors are highly data-dependent.

## 4.2. Data service

Several data service choices were provided for a mobile device to connect to the outside world. To select the preferred one for this journey planner, first the criteria for comparison will be set and base on them each choice will be evaluated.

### 4.2.1. Criteria for comparison

***Availability*** – One of the most important criteria is what percentage of mobile devices the data service can be used on.

***Ease of use*** – It is important whether the user can understand how the service works. This can be easily forgotten by the developer, because he can spend a lot of time around his product, understanding it perfectly, and forgetting how the first time user will see it. People are very diverse in all their aspects, especially in their technical understanding. Some can grasp the workings of a tool immediately, many others can get stuck on some technical quirk. If it is not obvious how the service works, many give up on the service, thinking it is not worth the trouble. When user requires directions, not to mention when he is on the move with the mobile phone, he does not have the time to study the tool. Even when there is time to study it, users are not known for reading manuals.

This chapter is about the data service used. But ease of use would seem to apply mainly for the user interface, the presentation layer of the tool. But the data service selected often cannot be completely hidden for the user. For example, when using the web page service, user is aware that that there is an additional requirement of being connected to the Internet to use the service. Another difficulty is the price of using the service. Even though these examples are overlapping with the criteria of availability and price, in the ease of use they have their influence on how complex or difficult the tool seems. That is why a structure of the tool is important.

***Initial cost of obtaining the data service*** – First of the criteria covering money, this one includes the cost of making the service available at all from the developer side. From the user perspective the data service is already available, the initial cost is none. One way or another, the service will eventually reach an Internet server that is common to all the data services, so the cost of setting up a server with a tool is also ignored, only the data service alone is considered.

***Per use cost of the data service*** – Includes the cost of a single journey query from both the user and the developer perspective. It may vary between service providers.

### 4.2.2. Evaluation

***SMS service (presentation layer)***

*Availability* – An SMS service is available on virtually every cell phone and smartphone. On pure PDAs it is still possible to send an SMS through some Internet applications, like ICQ. In relation to other services, SMS can be viewed as the universally available basic function.

*Ease of use* – Great advantage of SMS messaging universality is that almost everybody can write an SMS message without a problem, with no training necessary. The trouble is the format of a message. A tool on the other side has to parse the information to interpret individual parameters of a query and that requires a specific encoding. A very low portion of users are patient enough to learn the encoding. It may seem unlikely, but many users are also confused by the simplest encoding types and conventions, which a programmer learns even during his first experience with command line arguments. Last significant problem is the availability of help or a manual to teach such encoding. When using only an SMS service, a mobile device will not have a good way to explain it and all will depend on external information.

*Initial cost of obtaining the data service* – The developer has to use the SMS gateway service. The initial price consist mainly of renting a phone number, which can cost at most 100 USD for setup, and 25 USD as monthly fees, but can be acquired cheaper, it depends on the provider.

*Per use cost of the data service* – One outgoing SMS message costs on average 0.11 USD [23]. Service providers usually offer discount plans bringing the price lower, without such plans price per SMS is mostly between 0.10 – 0.20 USD. In addition there is a cost per message at the SMS gateway. For commercial gateway it can be around 0.015 USD [24], an alternative is to use your own gateway, such as an open source project Kannel [25].

A reply message can cost around 0.05 USD [24], or in case of own gateway at most a minor switching fee.

***Email (presentation layer)***

*Availability* – Unlike SMS, email is not available in pure cell phones, it is limited to devices with more program functions, PDAs and smartphones.
*Ease of use* – Email provides more freedom than an SMS, allowing longer texts. Problems of encoding persist, but mobile devices supporting email generally provide better ways to help and explain the tool. Additional concern is the need for an active Internet connection, as opposed to ever present SMS connection.
*Initial cost of obtaining the data service* – Registering an email address, which is either free or for a small fee.
*Per use cost of the data service* – Cost per email is tied to the cost of data used, depends on the provider, usually much cheaper than an SMS message.

***Web page connection (presentation layer)***
*Availability* – Like email, it is not available on pure cellphones, but mostly on PDAs and smartphones. Availability of a web page is strongly tied with email (presentation layer). When a web page is accessible, email is too, but not the other way around.
*Ease of use* – Web browsing allows to present a user friendly interface, making it easy to enter search parameters. Active Internet connection is needed.
*Initial cost of obtaining the data service* – Registering a domain name for a small fee.
*Per use cost of the data service* – Cost is tied to the cost of data used, depends on the provider. With more advanced machines it is cheaper than for a limited WAP browser.

***Internet connection (internal communication)***
*Availability* – The mobile device must support part of the tool. When a mobile device supports part of the tool and sending emails, it is bound to support IP connection as well.
*Ease of use* – It is hidden from the user completely, apart from the need for an active Internet connection. Lacks the restriction of an email format, making it the most flexible way to share information between the tool parts.
*Initial cost of obtaining the data service* – A possible fee for having part of the tool on the mobile device.
*Per use cost of the data service* – Cost is tied entirely to the cost of data used, depends on the provider and a purchased data plan.

***No data service***

*Availability* – Could be located only on mobile devices supporting more complex applications, mainly several megabytes of memory for data storage.

*Ease of use* – The tool could be customized to the device, providing any description and help necessary. No limitation is present by data services. However, it will be shown later that the tool might work with dynamic information, not to mention that transit schedules change periodically. To update the tool's data user would have to reinstall or update the tool manually or rely on some data service anyway to supply new data. This limits the tools capabilities and creates the risk of out of date results.

*Initial cost of obtaining the data service* – A possible fee for having the tool on the mobile device.

*Per use cost of the data service* – None.

### 4.2.3. Interpretation

After analyzing advantages and disadvantages of several solutions, they can be compared as a whole. When developing a tool, it is possible to choose just one approach and use it, or combine several approaches for better universality. Single solutions appear suitable as follows:

1. ***Internet connection –*** Provides the best combination of availability and ease of use., despite the necessity of installing a part of the tool on a mobile device.
2. ***Web page connection*** – Closely second, provides only a little bit less availability with similar ease of use. It is also the easiest to implement.
3. ***SMS service*** – Leading advantage of this solution is its availability. The most universal solution of all lacks mainly a user base patient enough to understand and use it.
4. ***Email*** – Has slightly different availability compared to an SMS service, but in the end covers much less number of mobile devices. Suffers all its disadvantages apart from being a little cheaper.
5. ***No data service*** – Compared to the Internet connection solution, the constraints on updated data and dynamic features together with increased hardware requirements outweigh the freedom from data services. Falls to SMS and email in much reduced

availability because of the hardware requirements.

The methods are not mutually exclusive, the tool can incorporate several of them, making it more universal and flexible. When a part of the tool is located on the Internet, several data services can connect mobile devices to the same server. Therefore it is purely on the developer how sophisticated he wants the tool to be. When choosing multiple solutions together it is desirable for them to complement each others weak points. For this purpose it is recommended to combine either web page or the Internet connection with an SMS service. Together they provide the greatest availability, web page's or the Internet connection's ease of use with SMS coverage as a backup. However, because of the additional technical difficulties during implementation and confusion of the user, it is better to choose just one data service.
Since the Internet connection was evaluated as the preferred data service, it will be used in this journey planner.

## 4.3. Architecture

The sizes of files containing the GTFS data vary from few hundreds of kilobytes to tens of megabytes. Such amount cannot be stored on low-end devices. The best way to make the tool more universal is to separate it to two parts, on on the client-side and one on the server side. The data service chosen is the Internet connection, which can connect the two parts. As for the logic, there is the presentation layer, the search algorithm core, the storage of traffic data and storage of data preprocessed for the algorithm. Since the mobile device part is the one with resource constraints, it only has to house the presentation layer. The rest of the application can be located on the server, where it will have access to much higher computing power and more memory, enough to contain all the GTFS data in the main memory without relying on a database.
If the journey planner was a commercial application, it would make sense to provide several choices for the user, as was said when choosing the data service. The presentation layer could be a web page, without the need for a custom mobile application. The other option for high-end users could involve a standalone mobile application which would connect to the outside world only for transport data updates. The mobile application part could also house part of the engine logic, to allow partial searches when offline. But for the purpose of this work, the most

universal approach was chosen.

## 4.4. Programming language

The programming language of the server is irrelevant, it does not depend on the mobile device limitations, the main choice is the mobile device part language. When choosing a programming language what matters most is the scope of the tool. If it is a little project the choice of Java ME is obvious. It is portable and third party applications are easy to use on that platform. But when the tool is supposed to be commercially successful and widely available, specific languages do not matter. The company developing such application will try to cover as wide range of devices as possible, using whatever language is available at each one. Good example is making an application for iPhone OS. It would be written in the native language and distribution agreements would be handled at corporate level. Since the goal is to provide an interface for a specific environment, Java ME was chosen. To complement this choice on the server side, standard Java was chosen there.

## 4.5. Algorithm

With the architecture decided, the algorithm can be chosen, to be on a server without mobile restrains.

### 4.5.1. Criteria

1.Memory requirements – How much does the memory required increase with the size and density of a traffic network. Algorithms generally have two main phases. The first one is the pre-processing. That is done only once, before any queries are made, independent on the actual query computation. The second is a single query computation, taking advantage of the data prepared in the first phase. Memory requirements will be considered separately for each phase. To define variables to tie the memory requirement to:

$N$ - The number of stops in the network.

$P$ - The number of vehicle runs in the network.

$\overline{P}$ - The average number of vehicles to pass through a stop.

$\overline{N}$ – The average number of stops on a vehicle run.

The data model of a transport network differs between algorithms and source formats, so it will be included later during implementation based on choices made.

2.Speed – How much does the speed of a query increase with the size and density of a traffic network. The main concern is the speed of the second, single query phase. Speed of the pre-processing is irrelevant for the user.

3.Support for additional search parameters – Other than the mandatory parameters like time and places of the trip.

4.Precision – Algorithms return either the optimal or suboptimal result.

### 4.5.2. Evaluation

1.The headway algorithms are a more general class of algorithms, that can have various memory requirements, speed and parameter support. But their precision depends largely on the transport network data. When the transport data would be supplied in form of headway estimations, such algorithm would be a natural choice. However, the data this application will work with will contain specific schedules. A headway algorithm would be viable only if the transport was so unreliable, that the specific schedules would not mean much more than hinting about the frequency of services.
2. The brute force use of a search algorithm like Dijkstra [19] would make it an adjacent node search with time as a boundary between open and closed nodes. For pre-processing the only thing to do is to load the network in nodes and in them store departures and arrivals, no extra pre-processing and memory is required. The speed using Fibonacci Heaps is the usual speed of Dijkstra $O(P*\overline{N} + NlogN)$, where $P*\overline{N}$ is the number of edges in the network. This includes only the basic search parameters, adding minimization of other parameters would decrease the speed. For example when minimizing the number of transfers, the search would have to remember best path reached on a node for every possible number of transfers. The supported parameters are therefore the basic ones, with possible augmentations to include more search parameters, with increased complexity. This is not an approximation, so the precision is optimal.

3. The algorithm based on transfer matrices requires a large amount of space for preprocessed data $O(N^2*maxt)$, where *maxt* is the maximum number of transfers. The speed of pre-processing is also $O(N^2*maxt)$. The asymptotic speed of the query is irrelevant, since its branching of recursion can be theoretically exponential. But the pre-processing and approximation make this algorithm fast. Additional search parameter support include number of transfers minimization. But in the end algorithm is still only an approximation.
4.The labeling algorithm presented requires memory for minimal distance matrix of $O(N^2)$ and time to compute it $O(N^2)$. The query time of the graph search is $O(P*\overline{N} + NlogN)$, and the additional search parameter is the slight minimization of number of transfers by preference during search approximation. The approximation might make better average speeds, but it still returns suboptimal results.
5. The pre-processing does not require any additional memory. Precision optimal for travel time. The speed of a query is $O(P*\overline{N})$. This algorithm is interesting in providing a suitable data model and reducing the N and P by selecting only a relevant portion of the network. The precision is optimal.
6.The pre-processing of this algorithm does not require any additional memory than that is required by standard loading of the network. The query uses recursion, but states that the algorithm speed does not increase significantly by network, size. So the asymptotic estimate would be misleading. The reasonable path minimizes travel time, number of transfers and walking time. However the precision is suboptimal.

### 4.5.3. Choice

Based on the properties of the algorithms, the pattern-first search [20] was selected. The lack of additional search parameters does not seem more important for a mobile user that the speed and precision of a basic search.

## 4.6. Search parameters

The algorithm is selected so it is time to evaluate which search parameters is should support.
The basic parameters of departure and destination are required for the algorithm. The time

parameter can be either a departure or an arrival, the user can choose. The algorithm has two version, forward and backward, searching either from the departure forwards in time, or from the arrival backwards in time.

This algorithm does not include the minimization of the number of exchanges, however it does support the limits of exchange duration. The exchange location constraints can be implemented as well.

Walking between stations is supported, but only partly, as possible transfers are hidden inside a transfer node, but this functionality can be added, along with parameters to influence it. More on this topic will be covered in the implementation chapter.

The reliability can be supported only in restrictions on individual transfers. The minimum reliability of each exchange on the trip, the number of travel minutes to sacrifice for a percent in reliability of one exchange and the number of minutes it takes for an exchange to be considered safe, all those can be implemented by placing restrictions during transfer computation. They also require the additional reliability data, like the common delay.

To minimize the travel duration in given time interval is also not supported as it would require many consecutive searches.

Transit specific properties of individual transport companies are hidden by the universal standard of the GTFS, so for their support there is no traffic data. The only exception is the information about fares, which is included in the GTFS standard, but not supported by the algorithm.

Abilities of existent journey planners showed that only a few basic search parameters are necessary. When a journey planner provides additional criteria, it is mainly because it has resources to do so, having transport data specific enough to implement them. The fact that additional parameters are optional is clear form the abilities of Google Transit, which is able to gain popularity without them. One may even say that its simplicity is an advantage.

# 5. Implementation

By following the design, the journey planner was implemented. In this chapter the implementation will be described, showing the approaches to individual problems and implementation details.

## 5.1. Transit data parsing

When the journey planner is started, the first thing it has to do is load the transit data. The chosen data source format is the GTFS, that comes in a set of files containing data in a certain model. However, the data model used by the algorithm is different in certain aspects, so some compromises needed to be made.

### 5.1.1. The GTFS data model

The GTFS standard is organized as follows:
Stops – Contained in the file „stops.txt“, includes information about places that pick up and drop off passengers. It is the basic location primitive used, which as a bonus contains mandatory location information.
Transfers – A journey planner is expected to derive possible transfers by walking by a location proximity. The file „transfers.txt“ contains information that complements the proximity estimation by recommending some transfers, providing minimal walking time or excluding transfers that are not possible.
Routes – Contained in „routes.txt“, a route is a basic primitive for a single transport line. A route is not defined as a sequence of stops, that is defined elsewhere. The only information relevant for this journey planner is the name of the route.
Trips – In „trips.txt“, it contains trip definitions. Multiple trips can belong to a single route/service, and have their own stop sequence and time point information, that is at what time at each stop this trip arrives and departs. In this file however, only trip definition is included, stop sequences and time point information are included in „stop_times.txt“. For each stop of a trip, there is an entry with departure and arrival times at that stop. By parsing

the entries, a complete stop sequence and time point information of a trip can be pieced together.

Services – The time information is defined in time of day. A service completes the time information of trips by providing a set of dates in a calendar when it is active. In file „calendar.txt“ services have their weekly validity defined, for example if a service is active at weekdays or weekends. This is done by a yes or no value at each day of the week. In „calendar_dates.txt“ there is a set of individual dates. In case both files are present, the individual days provide exceptions in the weekly schedule and additions.

Frequencies – A trip usually provides information about one vehicle's schedule in the day. By providing information in „frequencies.txt“, information about several trips can be grouped together, by setting the start time of the frequency, the end time and a headway. This is equal to several consecutive trips on a same stop sequence running after each other after a same time interval (the headway).

To make it more intuitive, the data model will be shown on an example. There is a tram line number 12. The tram line 12 is the route. Each vehicle in a day with this number has its own corresponding trip, different trips are also for different directions. So a tram number 12 that departs at 14:00 at one station X has a different trip defined than the next tram number 12 that departs from the same station X at 14:10. The only way these two vehicles can share a same trip is for the trip to be a defined as a frequency, with a 10 minute headway. The two directions can share one route, or route can be defined for each direction, it is up to the transit company. If the tram 12 has different schedules for weekdays, weekends or Fridays, then for each different set of days there will be a separate set of trips. Such set will have a service associated with them, telling which set of days this schedule is valid for.

### 5.1.2. The internal data model of the pattern-first search

The data model used by the pattern-first search algorithm[20] is defined in [26] . In general it is divided in two parts, static and dynamic. The static part in its basics looks as follows:

Stops – It is the same location primitive as in the GTFS standard.

Patterns – A pattern is a stop sequence in one direction. When compared with GTFS, it could be understood as a collection of trips on the same stop sequence. This grouping is more general than the frequency grouping in GTFS. The time validity of a pattern is not defined as

specifically as in GTFS by services, but by general information when this pattern is „active“.
Transfer nodes – A transfer node is a collection of stops sharing their location and logical meaning. For example when multiple stops belong to a location like a square or a road intersection, all the stops there belong to the same transfer node. In addition, transfer nodes a re only those set of stops, where it is possible to make a transfer from one pattern to another.
Creation of this static network is part of the algorithm pre-processing. When a query is received, there should be an active network created:
Active patterns – An active pattern unlike a pattern does not contain a sequence of stops, but only a set of transfer nodes. The time information about patterns is used in generation of active patterns so that only patterns relevant to the time specified in a query are converted to active ones.
Active transfer nodes – A subset of transfer nodes that are used for transfers between active patterns. Also an origin and destination of the trip are converted to active patterns.

By creating this „active network“, the search has to traverse much smaller amount of information.

### 5.1.3. The algorithm's internal data model

The algorithm used in the journey planner has to use a data model that enables the use of the algorithm, but does not loose too much information in the conversion from the GTFS. The following data model was chosen:
Routes – From GTFS, contains only the name of the route for result display.
Stops – From both models, stops are the same, so they are used without modification as a basic location primitive. Contains location coordinates.
Trips – From GTFS, a trip has a stop sequence and belongs either to a single vehicle on a route or to a group of vehicles as defined in GTFS frequency.
Transfer nodes – This is a compromise. In GTFS there is an option to group stops into stations. When this grouping is used in the source data, the stops are grouped to this single transfer node. Otherwise, each stop has its own corresponding node, regardless on the transferability on it. This greatly increases the network searched, since in pattern-first search only a fraction of stops belong to a transfer node. Also stops sharing the same name and close

location share the same transfer node.

Pattern – Each trip has its corresponding pattern. The frequency grouping of a GTFS trips is the only vehicle grouping a pattern gets. Unlike original patterns, here the patterns already contain the sequence of transfer nodes, instead of stops. Every stop has its transfer node, because unlike patterns rarely sharing a stop sequence, many trips do. Therefore it is possible to transfer from a trip to another trip going the same way. GTFS includes the possibility, that the trips overtake each other. Excluding stops from being a transfer node could eliminate some journey choices, making the result possibly suboptimal.

Services – Each trip and therefore pattern has its availability defined by the GTFS services.

Transfers – The full fledged transfer nodes would include transfers internally, ignoring the finer specification of transferring. Since transfer nodes used here only group stations, transfers from GTFS can be used. The transfers are computed like they are supposed to from GTFS, by walking distance with additions and exceptions of GTFS transfer information.

This concludes the data loading and pre-processing. The next part will be processing after receiving a single query, similar to generating an active network.

There are no active patterns or active transfer nodes. All the transfer nodes are considered active and are already prepared for their use in the algorithm. Selecting only a portion of them would not decrease their number much, but would require additional processing. The patterns however are at least filtered. Since patterns used already contain sequence of transfer nodes, they are prepared for their use in the algorithm and do not need to be converted. The only thing algorithm has to do is to select those patterns that are active in the time of the query. This is done by comparing their corresponding service with the time interval that is being explored.

The data model chosen makes a compromise between the more detailed GTFS one, and the faster pattern-first one, leaning more to the GTFS, to preserve precision over speed.

## 5.2. The algorithm

### 5.2.1. The algorithm's basic search

The pattern-first algorithm [20] was adapted for use on the modified data model. By having

patterns and transfer nodes like in the original model, the algorithm does not change much.

When a query is received and a corresponding active network generated, the first step is to initialize the transfer nodes. In essence, each node remembers last step of the best path found so far from the origin to it. Mainly the arrival time into it, which is later used by the algorithm to select the next node to explore. The origin is initialized with the departure information, and the rest of the nodes are initialized with not yet having any best path to them.

The search loop itself chooses an open node with the earliest best arrival time to it and „relaxes“ all the outgoing patterns. Then it closes the node. After the destination is chosen to relax, the algorithm ends and the best path is pieced from the steps from the destination.

This looks like a simple graph search algorithm, that uses the best arrival time boundary between the closed and open nodes, similar to Dijkstra [19]. But lets have a look at the differences. The main difference is not the logic of the algorithm, but the way it handles the data, for success of this algorithm stands on its data model. Unlike a adjacent-node search, it ignores neighbors and handles whole patterns, taking advantage of the nature of transit networks. The quick selection of active patterns is the advantage. If the data would support it, the grouping of many vehicle runs in a single pattern would reduce the amount of data to search through. This advantage is still present in form of frequency grouping, so its usefulness is data-dependent. When patterns are relaxed, the best arriving pattern so far is ignored, and the rest are updated on all their remaining forward length. That saves the algorithm from updating them later.

The complexity of choosing the node for a search step is $O(NlogN)$, since binomial heap is used. It could be optionally improved by Fibonacci heap to $N$. Each pattern can be updated by at most the number of nodes on that pattern, so the complexity of pattern updating is $O(P*\overline{N})$ which is equal to $O(N*\overline{P})$. Each node can have its key in the binomial heap decreased during pattern updating by every pattern going through it, but together all those decreases of a key equal to a single decrease, since the key is never increased. So the complexity of that step is $O(NlogN)$. Therefore the total complexity of a query is $O(N(logN+ \overline{P}))$. To recapitulate, $N$ is the number of transfer nodes, $\overline{N}$ is the average number of transfer nodes on a pattern, $P$ is the number of active patterns and $\overline{P}$ is the average number of active patterns on a node.

### 5.2.2. Trip duration minimization

The previously described search presumes that the minimal departure time is selected, and the user looks for an earliest arrival. An opposite task is also available, to find an earliest departure with the maximum arrival time specified. The algorithm is only mirrored in time.

Since the speed of the algorithm depends on how many patterns it has to search through, the algorithm does not just run a single search for one query. It starts by searching for a small time interval, by default two hours. It searches in the interval from departure time to departure time plus two hours, or arrival time minus two hours to arrival time. If a result is not found, the search is run again with the interval doubled. This increase is done until the interval reaches a full day, when the algorithm gives up, returning no result found. Since most of the journeys do not take more than two hours to complete, this approach saves time in most cases, increases time in rare ones, while it still supports searching for long trips.

When a journey is found, it provides the best path to the destination. But it does not minimize the duration. It follows a hungry approach, providing the earliest departures or latest arrivals at all stops on the way. To illustrate it on an example, there is a path from *A* to *B* to *C*. There is a hourly connection from *A* to *B*, but only a single evening connection from *B* to *C*. When using this algorithm in the morning, it would find the earliest morning departure to *B*. Then the user would have to wait whole day at *B* for an evening connection to *C*. When also minimizing the trip duration, the result would return a connection from *A* to *B* in the evening, just before the connection from *B* to *C* would depart.

While user might prefer the earliest possible connections to reduce the chance of a missed transfer, generally the trip duration should be minimized. To solve this problem, the algorithm takes advantage of its two mirror versions and runs them both. The first provides the best target time and the second opposite search is run from the target time backwards to minimize trip duration.

### 5.2.3. A* consideration

With the mandatory location information in the GTFS format, there was a possibility to include A* heuristics in the graph search. A* works by including a minimal possible distance to the destination in the weight of nodes. When the search works with time boundary the

minimal distance to the destination must be in form of time, that is the minimal amount of time it could possibly take to reach the destination. To compute this minimum, the straight distance to the destination must be divided by the maximum speed achieved on the network. Any slower speed could lead to suboptimal results. When the search would traverse faster connections in the direction of the destination, time weight would decrease. That would break rules on which the algorithm relies on. An algorithm could prematurely close a node through which an optimal journey would go.
The usability of A* therefore depends on the maximum speed achieved on the network. That is easy to compute during data loading, but experiments have shown that it is often an infinite speed or close to it. All it takes is a single connection close enough to have the same arrival and departure time specified, and the speed on this link is infinite. Based on these findings, A* heuristics was not implemented.

### 5.2.4. Supported search parameters

The search algorithm focuses on speed and precision, sacrificing the support of additional search parameters and minimization. Still it supports some of them, even though they are not presented to the user in the mobile interface to simplify it.
The first is the maximum walking distance, used in the initial data loading. If presented to the user, all it would take to add it would be to recompute transfers in transfer nodes for each query. The current maximum walking distance is set to a default 500 meters.
The second is the average walking speed during transfers. That is ready to be set without any modification necessary, now set by a constant to one meter per second.
The last is the minimum amount of time for a transfer, set also by default to two minutes.

## 5.3. Server structure

When the server is started, it enters the command loop, by which it can be controlled from a console by commands. To load the data there is a „load“ command, that prepares an algorithm instance for processing queries. When this is done, the server can be turned on for clients by the „start“ command, that creates a client request receiver is a separate thread, listening for clients.

An IP connection is used between the client and the server. However the only mandatory connection support in a MIDP 1.0 of the Java ME is the Http connection. To make the application universal, the connection is made using a Http protocol. The client request receiver implements a basic Http server listening on the local port 80, creating a new thread for every incoming client request. A request is handled as a POST request expecting predefined data content used only by the client side part of the journey planner. Any other request on this server will result in a reply saying that this server is used only for a journey planner mobile application. The data part of the POST request contains encoded journey planner query, that is decoded and supplied to the algorithm instance created during data loading. The result is encoded in a Http reply to the mobile application and sent back.

All the threads share a single algorithm instance, which is limited to one call at a time. That is because the algorithm runs on unsynchronized data structures to increase speed. For this implementation it as not necessary to create a server capable of handling a heavy load. That can be easily achieved by having more instances of the algorithm at a time, increasing the memory usage significantly. Another approach would be to run the algorithm on synchronized data structures, that would support multiple queries at a time, but the synchronization would lead to increased computation time. So for now when two clients request access to the algorithm, one of them waits, until the other one finishes. However, considering the high speed of a query, it would take a significant load for the requests to start piling up even for this implementation.

The command loop is used for simple server control and debugging purposes. It can place its own query and provides commands to safely shut down the server, ending the request receiver thread that is listening for client requests.

## 5.4. Mobile device part

The mobile device part is a midlet containing the presentation layer. Under it is a functionality that encodes a query into a POST request, sends it to the server and waits for a reply. A query uses a single Http connection, that is closed after the reply is received.

Java ME provides universal tools for designers in MIDP to use when creating a window. The implementation of these tools is left to the device manufacturer. So when creating a window the designer specifies entities like text fields, date fields and responses to universal

commands, without having to worry about screen size or button positions.

The interface of the journey planner consists of three screens, the first one is the input screen, where user selects the origin of a trip, the destination and time. Under those fields there is a choice field if the specified time should be interpreted as an arrival time or a departure time.

If the user enters the information and submits it, the application switches to the second screen, used for waiting. This screen will make the user know that the query was submitted and the application is waiting for a result. When client receives a reply or an error was encountered, the waiting screen switches to the results screen, containing the text form of the result or an error message. The user can either exit the midlet, or return to the input page to modify the query.

Before the midlet is compiled and deployed, it is necessary to select the location of the server either as an IP address or a DNS address, that is to be hardwired into the midlet. For debugging purposes the current value where to look for the server is localhost.

The hardware requirements of the mobile application are negligible, as it is written in the minimalistic manner for universality. After compilation the resulting jad and jar files have less than 9 kilobytes, using functionality of java that fits in a library the size of 38 kilobytes.

## 5.5. Measurement

The measurements of the server-side applications are presented in the following tables. The table number 1 shows the statistics of the measured networks. The number of transfer nodes is smaller than the number of stops, because some stops are aggregated under one transfer node. Stops are aggregated if they are defined as part of the same station, or if they have the same name while being close to each other.

(Table 1)

| Full network name | Stops | Transfer nodes | Patterns |
|---|---|---|---|
| Fredericton Transit | 667 | 614 | 134 |
| Milwaukee County Transit System | 5533 | 3470 | 9930 |
| Washington Metropolitan Area Transit Authority | 11646 | 8694 | 39851 |

The measured values are presented at table number 2. The measurement was done by placing 1000 queries. Each query has a randomly selected stops as origin and destination, the time of departure was randomly chosen from a period of one week. The first measured value is the loading time in seconds, which was done only once.

The second is the average number of search runs per a query. As was said above, a single query first tries to search in a small time interval of 2 hours for a journey. If it is not found, it makes another run with doubled time interval, until it reaches a whole day, after which it gives up. The average number of runs is the average number of those tries. The last search run backwards to minimize travel time is not included in the measurement.

A single search run is composed of the active network part generation, listed as pre-processing, and the graph search itself. Average values are given for both of those parts as well as for a whole run.

The last value is the average time of a query as a whole, including the backward search measurement. All the time values except the loading time are in milliseconds.

The measuring was done on a computer with processor Inter(R) Core(TM) i5 CPU, M430, 2.27GHz.

(Table 2)

| | Fredericton | Milwaukee | Washington |
|---|---|---|---|
| Loading time (s) | 0.25 | 79 | 486 |
| Average number of runs (ms) | 2.37 | 1.792 | 2.155 |
| Average pre-processing time (ms) | 0.0991 | 3.5117 | 14.8766 |
| Average graph search time (ms) | 0.4722 | 32.3398 | 97.9313 |
| Average whole run time (ms) | 0.5713 | 35.8516 | 112.8079 |
| Average whole query time (ms) | 1.715 | 79.607 | 297.601 |

The measured values show that this algorithm manages to efficiently find journeys even in large cities.

# 6. Conclusion

In this work we have looked at the aspects of journey planning on mobile devices.

First we studied the environment at which the application can be implemented, including the architecture, programming language and operating systems. Available data services for mobile devices were explored. Existing journey planners were discussed to show how other developers dealt with the design decisions.

The analysis continued by introducing transport data sources, with their formats and availability. A search algorithm possibilities were presented for the core of the journey planner, followed by analysis of possible search parameters.

In the design chapter all the information from analysis was evaluated and decisions were made for the form of the application. The application was implemented accordingly and described along with the additional decisions that arose.

The goal of this work was to develop a a search engine for optimal connection in a city mass public transport for mobile devices, which was accomplished. Search engine is independent on the operating system, since it is located on the server, that is independent on the environment of the mobile device. The interface for a specific environment was created for Java ME, a midlet runnable from the mobile device. The search is be able to work with static information, like schedules and walking distance, and also support dynamic information, like a common delay by including an interface to update the transit data on the fly, in case of common delay to manipulate the transfer time limitations.

The work explored the areas necessary to implement a journey planner and showed what it entails. Even though better alternatives arose for example in form of a much bigger project of Google Transit, the provided implementation is shown to be fast enough to be a successful demonstration of a journey planner development.

# References

All web page content referenced is as it was viewable 1 December 2010.

[1] Wikipedia (2010): PDA, <http://en.wikipedia.org/wiki/Personal_digital_assistant>.

[2] Oracle Corporation (2010): Java ME Technology, <http://java.sun.com/javame/technology>

[3] Gartner (2010): Worldwide Smartphone Sales to End Users by Operating System in 3Q10, <http://www.gartner.com/it/page.jsp?id=1466313>.

[4] Microsoft Windows Mobile operating systems, <http://www.microsoft.com/windowsmobile>.

[5] Microsoft Windows Embedded Studio development tools, <http://msdn2.microsoft.com/en-us/embedded>.

[6] Google Transit, <http://www.google.com/transit>.

[7] Transport Direct Portal, <http://www.transportdirect.info>.

[8] Idos, <http://jizdnirady.idnes.cz>.

[9] Smartřády, <http://www.smartrady.cz>.

[10] CEN TC278, Reference Data Model For Public Transport, EN12896.

[11] San Francisco Muni (2009): NextBus, <http://www.nextmuni.com>.

[12] Chicago Transit Authority (2010): CTA Bus Tracker, <http://www.ctabustracker.com>.

[13] Brian Ferris, Kari Watkins (2010): OneBusAway, <http://www.onebusaway.org/>.

[14] W.-T. Balke, W. Kießling, C. Unbehend (2004): Personalized Services for Mobile Route Planning: A Demonstration, *Proceedings 19th International Conference on Data Engineering.*

[15] Seungil Kim, Chungwon Lee, Youngchan Kim, Seungjae Lee, Dongjoo Park (2010): Error correction of arrival time prediction in real time bus information system, *Journal of Advanced Transportation,* 2010; 44:42–51.

[16] Bratislav Predic, Dragan Stojanovic, Slobodanka Djordjevic-Kajan, Aleksandar Milosavljevic, Dejan Rancic (2007): Prediction of Bus Motion and Continuous Query Processing for Traveler Information Services, *Faculty of Electronic Engineering, University of Nis, Serbia*.

[17] Dalia Tiesyte, Christian S. Jensen (2009): Assessing the Predictability of Scheduled-Vehicle Travel Times, *Department of Computer Science, Aalborg University, Denmark.*

[18] Jolanta Koszelew (2008): Two Methods of Quasi-Optimal Routes Generation in Public Transportation Network, *Faculty of Computer Science, Bialystok Technical University, Poland*.

[19] Dijkstra, E. W. (1959): A note on two problems in connexion with graphs. *Numerische Mathematik* **1**, 269–271.

[20] Ruihong Huang (2005): A Schedule-based Pathfinding Algorithm for Transit Networks Using Pattern First Search, *Geoinformatica* (2007) Volume 11, Number 2, 269-285.

[21] Man-chun Tan, C. O. Tong, S. C. Wong, Jian-min Xu (2007): An algorithm for finding reasonable paths in transit networks, *Journal of Advanced Transportation* (Autumn (Fall) 2007) Volume 41, Issue 3, pages 285–305.

[22] Mark D. Hickman, Nigel H. M. Wilson (1994): Passenger travel time and path choice implications of real-time transit information, *Transportation Research Part C: Emerging Technologies* (August 1995) Volume 3, Issue 4, Pages 211-226.

[23] Wikipedia (2010): SMS, <http://en.wikipedia.org/wiki/Mobile_phone>.

[24] Clickatell, <http://www.clickatell.com>.

[25] The Kannel Group (2009): Kannel: Open Source WAP and SMS gateway, <http://www.kannel.org>.

[26] Ruihong Huang, Z.-R. Peng (2008): A spatiotemporal data model for dynamic transit networks, *International Journal of Geographical Information Science* (May 2008) Vol. 22, No. 5, 527–545.

# CD Content

The enclosed CD contains:

*MassTransportRouting.pdf* – This text.

*InputDataLinks*

– *InputDataReferences.txt* – Contains references for transport data that could not have been included due to possible licence issues.

*Documentation*

– *Documentation.pdf* – Contains the guide how to use the journey planner.

– *ClientJavaDoc* – The JavaDoc of the client part.

– *ServerJavaDoc* – The JavaDoc of the server part.

*Project*

– *Searcher* – The NetBeans 6.7 project of the client containing source code.

– *SearcherServer* – The NetBeans 6.7 project of the server containing source code.

*Executable*

– *SearcherServer.jar* – The server side executable.